

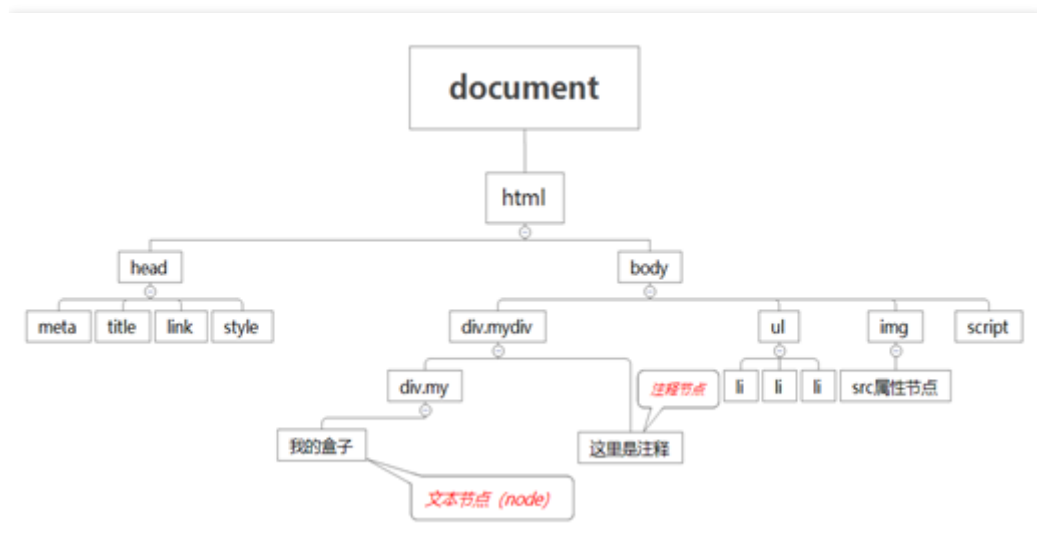
1、DOM概述:

全称Document Object Model文档对象模型

JS中通过DOM来对HTML文档进行操作，进行样式修改等

- 文档：整个HTML网页文档
- 对象：将网页中的每一个部分都转换为一个对象
- 模型：使用模型表示对象之间的关系，方便获取对象

DOM树:



2、节点概述:

Node: 是构成网页的基本组成部分，网页中的每一个部分都可以称之为节点。

EG: 标签、属性、文本、注释等；但分类也不同，类型不同所使用的属性和方法都不尽相同

- 标签：元素节点
- 属性：属性节点
- 文本：文本节点
- 文档：文档节点



节点的关系:

- 父子
- 兄弟
- 祖先

节点属性:

	nodeName	nodeType	nodeValue
文档节点	#document	9	null
元素节点	标签名	1	null
属性节点	属性名	2	属性值
文本节点	#text	3	★文本内容

```

1  <div class="mydiv">2</div><!-- 注释节点 -->
2  <div id="mydiv">id</div>
3  <script>
4      console.dir(document);
5      //nodeName: "#document" 文档节点
6      //nodeType: 9
7
8      console.dir(document.getElementById('mydiv'));
9      //nodeName: "DIV" 元素节点
10     //nodeType: 1
11
12     console.dir(document.getElementById('mydiv').childNodes);
13     //nodeName: "#text" 文本节点
14     //nodeType: 3
15
16     console.dir(document.getElementsByClassName('mydiv')[0].nextSibling)
17     //nodeName: "#comment" 注释节点
18     //nodeType: 8
19
20     console.log(document.querySelector('.mydiv').attributes)
21     //nodeName: "class" 属性节点
22     //nodeType: 2

```

```
▼ [[Prototype]]: Node
  ATTRIBUTE_NODE: 2
  CDATA_SECTION_NODE: 4
  COMMENT_NODE: 8
  DOCUMENT_FRAGMENT_NODE: 11
  DOCUMENT_NODE: 9
  DOCUMENT_POSITION_CONTAINED_BY: 16
  DOCUMENT_POSITION_CONTAINS: 8
  DOCUMENT_POSITION_DISCONNECTED: 1
  DOCUMENT_POSITION_FOLLOWING: 4
  DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC: 32
  DOCUMENT_POSITION_PRECEDING: 2
  DOCUMENT_TYPE_NODE: 10
  ELEMENT_NODE: 1
  ENTITY_NODE: 6
  ENTITY_REFERENCE_NODE: 5
  NOTATION_NODE: 12
  PROCESSING_INSTRUCTION_NODE: 7
  TEXT_NODE: 3
```

3、文档节点 (Document)

文档节点document，代表的是整个HTML文档，网页中的所有节点都是它的子节点

document对象作为window对象的属性存在的，我们不用获取可以直接使用

通过该对象我们可以在整个文档访问内查找节点对象，并可以通过该对象创建各种节点对象

1.document节点对象代表整个文档，每张网页都有自己的document对象。window.document属性就指向这个对象。

2.document对象的获取。

- 正常的网页，直接使用document或window.document。
- iframe框架里面的网页，使用iframe节点的contentDocument属性。
- Ajax 操作返回的文档，使用XMLHttpRequest对象的responseXML属性。
- 内部节点的ownerDocument属性。

3.对于 HTML 文档来说，document对象一般有两个子节点。

- 第一个子节点是document.doctype，指向<DOCTYPE>节点，即文档类型 (Document Type Declaration，简写DTD) 节点。document.firstChild通常就返回这个节点。
- 第二个子节点document.documentElement属性返回当前文档的根元素节点 (root) 。

4.document.body属性指向<body>节点。

5.document.head属性指向<head>节点。

6.document.forms属性返回所有<form>表单节点。

7.document.images属性返回页面所有图片节点。

8.document.embeds属性和document.plugins属性，都返回所有<embed>节点

9.document.scripts属性返回所有<script>节点。

10.document.styleSheets属性返回文档内嵌或引入的样式表集合(CSS)。

11.document.documentURI属性返回一个字符串，表示当前文档的网址。documentURI继承自Document接口，可用于所有文档。

12.document.URL属性都返回一个字符串，表示当前文档的网址。URL继承自HTMLDocument接口，只能用于HTML文档。

更多属性方法见：[\(2条消息\) 文档document节点，属性与方法_zdw火车叨位去的博客-CSDN博客](#)

4、元素节点 (Element)

HTML中的各种标签都是元素节点，这也是我们最常用的一个节点；浏览器会将页面中所有的标签都转换为一个元素节点，我们可以通过document的方法来获取元素节点

比如：

```
▶ getElementById: f getElementById()
▶ getElementsByClassName: f getElementsByClassName()
▶ getElementsByName: f getElementsByName()
▶ getElementsByTagName: f getElementsByTagName()

▶ querySelector: f querySelector()
▶ querySelectorAll: f querySelectorAll()
```

注意：DOM方法中除了getElementById和querySelector方法，其他方法获取的都是类数组
获取页面节点的方法：

1、使用内置的方法来获取：

- document.getElementById()：根据id属性值获取一个元素节点对象

```
1 <div id="mydiv">我的盒子</div>
2   <script>
3       let mydiv = document.getElementById('mydiv');
4       console.dir(mydiv);
5   </script>
```

```
▼ div#mydiv ⓘ
  accessKey: ""
  align: ""
  ariaAtomic: null
  ariaAutoComplete: null
  ariaBusy: null
  ariaChecked: null
  ariaColCount: null
  ariaColIndex: null
  ariaColSpan: null
  ariaCurrent: null
  ariaDescription: null
  ariaDisabled: null
  ariaExpanded: null
  ariaHasPopup: null
```

- document.getElementsByClassName(): 根据class属性值获取一个元素节点对象

```
1 <div class="mydiv">我的盒子</div>
2   <span class="my">poppoppop</span>
3   <ul class="mydiv">
4     <li>1</li>
5     <li>2</li>
6     <li>3</li>
7   </ul>
8   <script>
9     let mydiv1 = document.getElementsByClassName('mydiv');
10    console.dir(mydiv1);
11  </script>
```

```
▼ HTMLCollection(2) ⓘ
  ▶ 0: div.mydiv
  ▶ 1: ul.mydiv
  length: 2
  ▶ [[Prototype]]: HTMLCollection
```

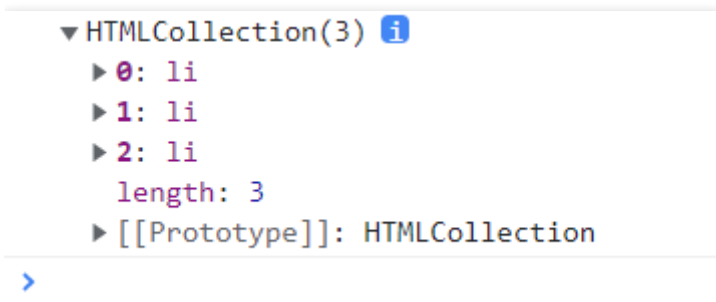
- document.getElementsByTagName: 根据标签获取节点, 返回的是一个类数组

```
1 <div class="mydiv">我的盒子</div>
2   <span class="my">poppoppop</span>
3   <ul class="mydiv">
```

```

4      <li>1</li>
5      <li>2</li>
6      <li>3</li>
7  </ul>
8  <script>
9      let lis = document.getElementsByTagName('li');
10     console.dir(lis);
11 </script>

```



- `document.getElementsByName`: 根据标签获取节点，返回的是一个类数组

```

1  <input type="text" name="username">
2  <input type="checkbox" name="hobby"> 学习
3  <input type="checkbox" name="hobby"> 运动
4  <input type="checkbox" name="hobby"> 打游戏
5  <script>
6      //返回类数组，即便只有一个元素
7      let input = document.getElementsByName('username');
8      console.log(input); //NodeList [input]
9
10
11     //返回类数组
12     let checkbox = document.getElementsByName('hobby');
13     console.log(checkbox); //NodeList(3) [input, input, input]
14 </script>

```

- `document.querySelector()`: 支持各种选择器，返回第一个满足条件的节点;
- `document.querySelectorAll()`: 返回所有的满足条件的节点

```

1  <input type="text" name="username">
2
3  <input type="checkbox" name="hobby"> 学习

```

```

4 <input type="checkbox" name="hobby"> 运动
5 <input type="checkbox" name="hobby"> 打游戏
6
7 <script>
8     //返回的是节点
9     let input = document.querySelector('input[name="username"]');
10    console.log(input); //<input type="text" name="username">
11
12    //Notice
13    let hobby = document.querySelector('input[name="hobby"]');
14    console.log(hobby); //<input type="checkbox" name="hobby">
15
16    //获取到所有的满足条件的节点
17    let hobbyArr = document.querySelectorAll('input[name="hobby"]');
18    console.log(hobbyArr); //0: input 1: input 2: input
19 </script>

```

2、根据节点的关系来获取：

```

1 <head>
2     <meta charset="UTF-8">
3     <meta http-equiv="X-UA-Compatible" content="IE=edge">
4     <meta name="viewport" content="width=device-width, initial-scale=1.0">
5     <title>document</title>
6     <style>
7         #mydiv {
8             width: 300px;
9             height: 50px;
10            background-color: #f01;
11            margin: 50px;
12        }
13    </style>
14 </head>
15
16 <body>
17     <div class="my">
18         独立的盒子
19     </div>
20

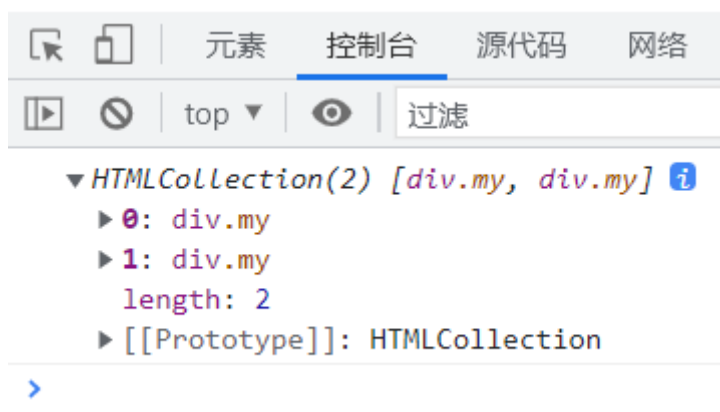
```

```

21     <div class="mydiv">
22         <div class="my">
23             第1个盒子
24         </div>
25         <div class="my">
26             第2个盒子
27         </div>
28     </div>
29
30     <script>
31         //注意下标
32         let mydiv = document.getElementsByClassName('mydiv')[0];
33         console.log(mydiv.getElementsByClassName('my'));
34         //HTMLCollection(2) [div.my, div.my]
35         mydiv.getElementsByClassName('my')[1].innerHTML = '666';
36     </script>
37 </body>

```

独立的盒子
第1个盒子
666



- 父节点: parentNode

```

1 <div class="mydiv">
2     <div class="my">
3         第1个盒子
4     </div>

```



```

5 </div>
6
7 <script>
8     let my1 = document.getElementsByClassName('my');
9     console.dir(my1);
10    console.dir(my1[0].parentNode); //获取父节点
11        //div.mydiv
12 </script>

```

```

▼ HTMLCollection(1) ⓘ
  ▶ 0: div.my
    length: 1
  ▶ [[Prototype]]: HTMLCollection
  ▶ div.mydiv
>

```

- 获取所有父级节点:

```

1 <div class="mydiv">
2     <div class="my">
3         <div class="span">
4             <div class="l11">
5                 100
6             </div>
7         </div>
8     </div>
9 </div>
10
11 <script>
12     let l11 = document.getElementsByClassName('l11')[0];
13     console.dir(l11);
14
15     Element.prototype.allParents = function () {
16         let arr = [];
17         let pNode = this.parentNode;
18         while (pNode && pNode != document) {
19             arr.push(pNode);
20             pNode = pNode.parentNode;
21         }
22         arr.pop();
23         arr.pop();

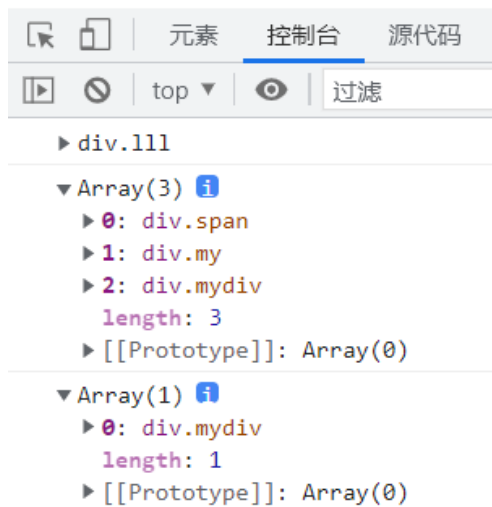
```

```

24         return arr;
25     }
26     console.log(l11.allParents());
27
28     console.log(document.getElementsByClassName('my')[0].allParents());
29     //['div.span', 'div.my', 'div.mydiv', 'body', 'html']
30 </script>

```

100



- 子: children child

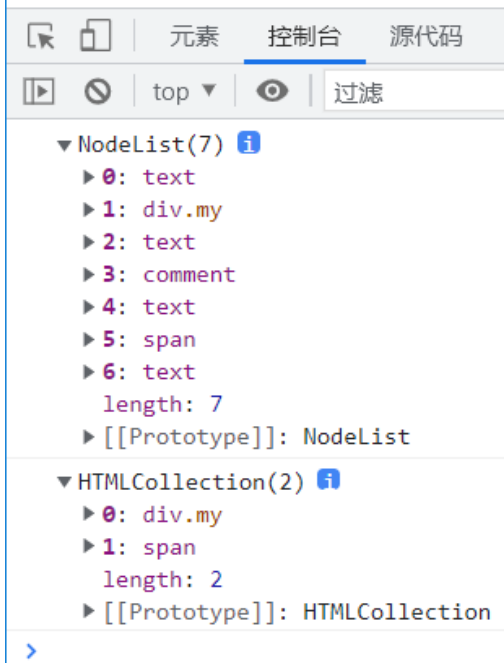
```

1  <div class="mydiv">
2      一个盒子
3      <div class="my">
4          儿子
5      </div>
6      <!-- 注释节点 -->
7      <span>
8          文本信息
9      </span>
10 </div>
11 <script>
12     let mydiv = document.getElementsByClassName('mydiv');
13     console.log(mydiv[0].childNodes);

```

```
14 //是个属性，获取的是子节点
15
16 console.log(mydiv[0].children);
17 //是个属性，获取的是子元素
18 </script>
```

一个盒子
儿子
文本信息



- 查找 HTML 父子:

方法	描述
元素节点.parentNode	返回元素的父节点。
元素节点.parentElement	返回元素的父元素。
元素节点.childNodes	返回元素的一个子节点的数组（包含空白文本Text节点）。
元素节点.children	返回元素的一个子元素的集合（不包含空白文本Text节点）。
元素节点.firstChild	返回元素的第一个子节点（包含空白文本Text节点）。
元素节点.firstElementChild	返回元素的第一个子元素（不包含空白文本Text节点）。
元素节点.lastChild	返回元素的最后一个子节点（包含空白文本Text节点）。
元素节点.lastElementChild	返回元素的最后一个子元素（不包含空白文本Text节点）。
元素节点.previousSibling	返回某个元素紧接之前节点（包含空白文本Text节点）。
元素节点.previousElementSibling	返回指定元素的前一个兄弟元素（相同节点树层中的前一个元素节点）。
元素节点.nextSibling	返回某个元素紧接之后节点（包含空白文本Text节点）。
元素节点.nextElementSibling	返回指定元素的后一个兄弟元素（相同节点树层中的下一个元素节点）。

```

1 <div>3</div>
2   <div class="mydiv">1</div>
3   <div>2</div>
4   <script>
5       let mydiv = document.getElementsByClassName('mydiv')[0];
6       console.dir(mydiv);
7
8       //下一个兄弟节点
9       console.log(mydiv.nextSibling);
10
11      //下一个兄弟元素
12      console.dir(mydiv.nextElementSibling);
13
14      //下一个兄弟元素 de 子节点
15      console.dir(mydiv.nextElementSibling.childNodes);
16
17      //上一个兄弟节点
18      console.log(1, mydiv.previousSibling);
19
20      //上一个兄弟元素

```

```

21     console.dir(mydiv.previousElementSibling);
22
23     //上一个兄弟元素 de 子节点
24     console.dir(mydiv.previousElementSibling.childNodes);
25 </script>

```

```

▶ div.mydiv
▶ #text
▶ div
▶ NodeList(1)
1 ▶ #text
▶ div
▶ NodeList(1)

```

- 节点关系: sibling.all

```

1     <div>3</div>
2     <div class="mydiv">2</div>
3     <div>1</div>
4     <ul>
5         <li></li>
6         <li></li>
7         <li></li>
8     </ul>
9     <a href="">连接</a>
10    <script>
11        let mydiv = document.getElementsByClassName('mydiv')[0];
12        Element.prototype.allSiblings = function () {
13            return [...this.parentNode.children];
14        }
15        console.log(mydiv.allSiblings()); //拿到所有的兄弟元素
16        //[div, div.mydiv, div, ul, a, script]
17
18        //用filter()过滤
19        Element.prototype.allSiblings = function () {
20            return [...this.parentNode.children].filter(e => e !== this);
21            //this指向的是距离他最近的function的调用者
22        }
23        console.log(mydiv.allSiblings()); //拿到所有的兄弟元素

```

```

24      //[div, div, ul, a, script]
25
26      //直接写成一个函数
27      function allElementSiblings(node) {
28          //this指向的是距离他最近的function的调用者
29          return [...node.parentNode.children].filter(e => e !== node);
30      }
31      console.log(allElementSiblings(mydiv));
32      //[div, div, ul, a, script]
33      </script>

```

- 更多节点获取方法：

```

1  <div>
2      <div>3</div>
3      <div class="mydiv">1</div>
4      <div>2</div>
5      <ul>
6          <li></li>
7          <li></li>
8          <li></li>
9      </ul>
10     <a href="">连接</a>
11     
12     
13     
14
15     <form action=""></form>
16     <form action=""></form>
17     <form action=""></form>
18 </div>
19
20
21 <div id="mydiv">
22     id
23 </div>
24 <script>
25     //快捷方式-->就是属性的熟悉，不算是知识点学习
26     console.dir(document);

```

```

27     console.log(document.body);
28     console.log(document.images);
29     console.log(document.forms);
30     console.log(document.getElementById('mydiv'));
31     //console.log(document.mydiv); //undefined
32 </script>

```

```

▶ #document
▶ <body>...</body>
▶ <script>...</script>
▶ HTMLCollection(3) [img, img, img]
▶ HTMLCollection(2) [form, form]
<div id="mydiv">id</div>

```

5、文本节点 (Text) :

文本节点表示的是HTML标签以外的文本内容，任意非HTML的文本都是文本节点

它包括可以字面解释的纯文本内容

文本节点一般是作为元素节点的子节点存在的

获取文本节点时，一般先要获取元素节点，再通过元素节点获取文本节点。例如：元素节点.firstChild;

获取元素节点的第一个子节点，一般为文本节点

1.nodeType的值为3。

2.nodeName的值为 "#text" 。

3.nodeValue或者的值为节点所包含的文本。

4.parentNode是一个Element。

5.appendData(text):将text添加到节点的末尾。

6.deleteData(offset, count): 从offset指定的位置开始删除count个字符。

7.insertData(offset,text):在offset指定的位置插入text。

8.replaceData(offset,count,text):用text替换从offset指定的位置开始到offset+count为止出的文本。

9.splitText(offset):从offset指定的位置将当前文本节点分成两个文本节点。

10.substringData(offset,count):提取从offset指定的位置开始到offse+count位置处的字符串。

11.createTextNode():创建文本节点。

12.normalize()合并文本节点。

13.splitText()分割文本节点。

原文链接: https://blog.csdn.net/weixin_28686771/article/details/111977451

6、属性节点 (Attr) :

属性节点表示的是标签中的一个一个的属性，这里要注意的是属性节点并非是元素节点的子节点，而是元素节点的一部分，可以通过元素节点来获取指定的属性节点。例如：元素节点.getAttributeNode("属性名"); 浏览器已经为我们提供文档节点对象，这个对象是window; 属性可以在页面中直接使用，文档节点代表的是整个网页。

```
1  setAttribute('属性名', '属性值') // 设置指定的属性名的值
2  getAttribute() // 获取指定的属性名的值 attributes属性获取所有
3  hasAttribute() // 检查一个属性是否存在
4  removeAttribute() // 删除指定的属性
5
6
7  <div data-id="6" name="'111" id="sss" class="ccc"></div>
8  var div = document.getElementsByTagName('div')[0];
9  // 1. 获取属性
10 console.log(div.attributes, div.attributes[0].value); // 获取所有属性 某个属性
11 console.log(div.getAttribute(name));
12 console.log(div.getAttribute('name'));
13
14 // 2. 设置属性(新增/修改)
15 div.attributes[0].value = '9'; // 修改属性成功
16 div.setAttribute('name', 'florence'); // 修改
17 div.setAttribute('data-arr', '333'); // 新增
18
19 // 3. 检测是否包含某个属性
20 console.log(div.hasAttribute('name')); // true
21
22 // 4. 删除某个属性
23 div.removeAttribute('name');
24 console.log(div.hasAttribute('name')); // false
25
26 <div class="mydiv" id="myv">22222</div>
27 <script>
28     let my = document.getElementById('myv');
29     console.log(my.attributes);
30
31     //万能的属性的操作:增(set)删(remove)改(set)查(get)
32     //用起来相对来讲比较复杂
33     my.setAttribute('data-id', 300); //增
```



```

34         my.setAttribute('mydiv', 'newvalue'); //改
35         my.removeAttribute('class'); //删除
36         console.log(my.getAttribute('id')); //查
37     </script>

```

```

▼ NamedNodeMap {0: class, 1: id, 2: myattr, class: class, id: id, myattr: myattr, Length: 3} ⓘ
  ▶ 0: id
  ▶ 1: myattr
  ▶ 2: data-id
  ▶ data-id: data-id
  ▶ id: id
  ▶ myattr: myattr
    length: 3
  ▶ [[Prototype]]: NamedNodeMap
myv

```

标准属性:

```

1  /*
2  1. A: href、target;
3  2. DIV: id;
4  3. img: src;
5  4. input的value;
6  5. checkbox的checked
7  6. ...
8  7. 标准属性可以直接使用: 节点.属性名; 如:节点.src = './topimg1.jpg';
9  **注意: class 属性不能直接操作**
10 */
11 <div class="mydiv" id="myv" myattr="myv">22222</div>
12 
13
14 <input type="text" value="300" name="username">
15 <script>
16     let my = document.getElementById('myv');
17     console.log(my.id);
18     console.log(document.querySelector('img').src);
19     document.querySelector('img').src
20     ='https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fimg.jj20.com%2Fup%2Fallimg%2F11
21     14%2F063021120F9%2F210630120F9-1-
22     1200.jpg&refer=http%3A%2F%2Fimg.jj20.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=a
23     uto?sec=1656142066&t=a3a8b36573cd1666beb7df4571715a5d';
24
25     console.dir(document.querySelector('input[name="username"]'));
26     document.querySelector('input[name="username"]').value = '蓝鲸';
27 </script>

```

class属性:

```
1 <div class="mydiv p o i my">22222</div>
2 <script>
3     let my = document.getElementsByClassName('mydiv')[0];
4     console.log(my.className); //mydiv p o i my
5
6     //classList
7     console.dir(my); //div.mydiv.p.o.i.my
8     //增
9     my.classList.add('b');
10    console.log(my.className); //mydiv p o i my b
11    //删
12    my.classList.remove('i');
13    console.log(my.className); //mydiv p o my b
14
15    //是否包含
16    console.log(my.classList.contains('p')); //true
17    console.log(my.classList.contains('my')); //true
18    console.log(my.classList.contains('my1')); //false
19
20    //toggle:有的删除，没有的追加
21    my.classList.toggle('mydiv');
22    console.log(my.className); //p o my b
23    my.classList.toggle('gg');
24    console.log(my.className); //p o my b gg
25 </script>
```

```
1 // 一次性修改元素的classList
2 el.className;
3
4 // classList: 节点的classList提供了操作class的更专业的方法:
5 classList.add() // 在不影响原始class的情况下追加新的class;
6 classList.remove() // 删除指定的class, 没指定的不会删除;
7 classList.contains() // 检查是否存在指定的class;
8 classList.replace() // 将前者类名替换为后者类名
9
```

```
10 <div class="name1 name2"></div>
11 <script>
12
13     var div = document.getElementsByTagName('div')[0];
14     console.log(div.className); // 1. 获取
15     console.log(div.classList); // 1. 获取
16
17     div.className = 'www mmm sss'; // 2. 修改
18     div.classList.replace('rrr', 'uuu'); // 2. 修改 将div的className为rrr的替换为uuu
19     div.classList[0] = 'sdf'; // 2. 修改不成功
20
21     div.className = 'www mmm'; // 3. 删除
22     div.classList.remove('www'); // 3. 删除
23
24     div.className = 'www mmm ddd'; // 4. 新增
25     div.classList.add('rrr'); // 4. 新增
26
27     console.log(div.classList.contains('rrr')); // 5. 检测 true
28     console.log(div.classList.contains('www')); // false
29 </script>
```

classList:

```

▼ classList: DOMTokenList(5)
  0: "mydiv"
  1: "p"
  2: "o"
  3: "i"
  4: "my"
  length: 5
  value: "mydiv p o i my"
  ▼ [[Prototype]]: DOMTokenList
    ▶ add: f add() 添加
    ▶ contains: f contains() 是否包含
    ▶ entries: f entries()
    ▶ forEach: f forEach()
    ▶ item: f item()
    ▶ keys: f keys()
    length: (...)
    ▶ remove: f remove() 移出指定的class
    ▶ replace: f replace()
    ▶ supports: f supports()
    ▶ toString: f toString()
    ▶ toggle: f toggle() 切换
    value: (...)
    ▶ values: f values()
    ▶ constructor: f DOMTokenList()
    ▶ Symbol(Symbol.iterator): f values()
    Symbol(Symbol.toStringTag): "DOMTokenList"
    ▶ get length: f length()
    ▶ get value: f value()
    ▶ set value: f value()
    ▶ [[Prototype]]: Object

```

自定义属性:

```

1  // 获取自定义属性的值, 例如标签的data-myattr = 'value'
2  el.dataset.myattr;
3
4  <div data-id="6"></div>
5  var div = document.getElementsByTagName('div')[0];
6  console.log(div.dataset.id); // 1. 获取
7  div.dataset.id = 9; // 2. 更改
8  div.dataset.name = 'Florence'; // 3. 新增
9  delete div.dataset.name; // 4. 删除
10
11  <div class="mydiv i my" data-uid="20" data-id="30">4545445</div>
12  <script>
13      let my = document.getElementsByClassName('mydiv')[0];
14      console.dir(my); //div.mydiv.i.my
15      console.dir(my.dataset.uid); //20
16      console.dir(my.dataset.id); //30

```

自定义属性：使用场景

```

1  <div>
2      <p>文章主题
3          <br>
4          文章标题
5          <br>
6          <div class="support" data-aid="31">点赞</div>
7          <div class="collect" data-aid="31">收藏</div>
8      </p>
9      <p>文章主题
10         <br>
11         文章标题
12         <br>
13         <div class="support" data-aid="31">点赞</div>
14         <div class="collect" data-aid="31">收藏</div>
15     </p>
16     <p>文章主题
17         <br>
18         文章标题
19         <br>
20         <div class="support" data-aid="31">点赞</div>
21         <div class="collect" data-aid="31">收藏</div>
22     </p>
23 </div>
24 <script>
25     let support = document.querySelectorAll('.support');
26     support.forEach(function (foo) {
27         foo.onclick = function () {
28             console.log(this.dataset.aid);
29         }
30     });
31 </script>

```

7、获取、修改HTML的值：

1. 获取HTML的值：

方法	描述
元素节点.innerHTML	获取 HTML 元素的 inner HTML
元素节点.innerText	获取 HTML 元素的 inner Text
元素节点.属性	获取 HTML 元素的属性值。
元素节点.getAttribute(attribute)	获取 HTML 元素的属性值。
元素节点.style.样式	获取 HTML 元素的行内样式值。
元素节点.src / href	获取 HTML 元素的连接地址

在其它浏览器中可以使用getComputedStyle()这个方法来获取元素当前的样式，这个方法是window的方法，可以直接使用，但是需要两个参数：

第一个参数：要获取样式的元素

第二个参数：可以传递一个伪元素，一般都传null

```
1 <head>
2     <meta charset="UTF-8">
3     <title></title>
4     <style>
5         /*样式表的样式*/
6         #box {
7             width: 200px;
8             height: 200px;
9             background-color: green;
10        }
11    </style>
12 </head>
13 <body>
14 <div style="width: 100px;height: 100px;" id="box"></div>
15
16 <script>
17     /*通用的获取元素样式的方法*/
18     function getStyle(obj, name) {
19         if (window.getComputedStyle) {
20             //正常浏览器的方式，具有getComputedStyle()方法
21             return getComputedStyle(obj, null)[name];
22         } else {
```

```

23         //IE8的方式，没有getComputedStyle()方法
24         return obj.currentStyle[name];
25     }
26 }
27
28 var box = document.getElementById("box");
29
30 console.log(getStyle(box, "width"));
31 console.log(getStyle(box, "height"));
32 console.log(getStyle(box, "background-color"));
33 </script>
34 </body>

```

编写一段兼容性代码，用来获取任意标签的文本内容：

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title></title>
6  </head>
7  <body>
8  <a href="https://www.baidu.com" id="a">打开百度，你就知道! </a>
9
10 <!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
11 <script>
12     var a = document.getElementById("a");
13
14     console.log(getInnerText(a));
15
16     /*获取任意标签的内容*/
17     function getInnerText(element) {
18         // 判断浏览器是否支持textContent,如果支持，则使用textContent获取内容，否则使用
19         innerText获取内容。
20         if(typeof element.textContent == "undefined") {
21             return element.innerText;
22         } else {
23             return element.textContent;
24         }
25     }

```

```
24     }
25 </script>
26 </body>
27 </html>
```

2. 改变 HTML 的值:

方法	描述
元素节点.innerHTML = <i>new text content</i>	改变元素的 inner Text。
元素节点.innerHTML = <i>new html content</i>	改变元素的 inner HTML。
元素节点.属性 = <i>new value</i>	改变 HTML 元素的属性值。
元素节点.setAttribute(<i>attribute</i> , <i>value</i>)	改变 HTML 元素的属性值。
元素节点.style.样式 = <i>new style</i>	改变 HTML 元素的行内样式值。

1 innerText

```
1 <button id="btn">我是按钮</button>
2
3 <!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
4 <script>
5     var btn = document.getElementById("btn");
6     btn.innerText = "我是JavaScript的按钮";
7     console.log(btn);
8 </script>
9
```

2 innerHTML

```
1 <div id="box"></div>
2 <script>
3     var box = document.getElementById("box");
4     box.innerHTML = "<h1>我是Box中的大标题</h1>";
5     console.log(box);
6
7     document.body.innerHTML += '<div>111</div>';
8     // 可以在页面中一一加入创建的元素，不会覆盖上一次添加的元素
9 </script>
10
```



```

11 // 包括内部html标签, 修改符为= 和 += 可对内容进行新增、修改、删除
12 el.innerHTML;
13 document.body.innerHTML += '<div>111</div>'; // 新增
14 document.body.innerHTML = ''; // 清空内容
15 document.body.innerHTML = '<li></li>'; // 修改/替换
16
17 // 包括内部html标签, 还包括自己
18 el.outerHTML
19
20 // 返回文本, 没有html标签, 赋值要小心
21 el.innerText
22

```

3

```

1 <a id="a" href="">打开百度, 你就知道! </a>
2
3 <script>
4     //1
5     var a = document.getElementById("a");
6     a.href="https://www.baidu.com";
7     console.log(a);
8
9     //2
10    var a = document.getElementById("a");
11    a.setAttribute("href", "https://www.baidu.com");
12    console.log(a);
13 </script>

```

3. 其他方法

修改节点的内容除了常用的innerHTML和innerText之外, 还有insertAdjacentHTML和insertAdjacentText方法, 可以在指定的地方插入内容。insertAdjacentText方法与insertAdjacentHTML方法类似, 只不过是插入纯文本, 参数相同。

语法说明:

```

1 object.insertAdjacentHTML(where,html);
2 object.insertAdjacentText(where,text)

```

参数说明:

1. where:

- beforeBegin: 插入到开始标签的前面
- beforeEnd: 插入到结束标签的前面
- afterBegin: 插入到开始标签的后面
- afterEnd: 插入到结束标签的后面



2. html: 一段html代码

3. text: 一段文本值

注意事项:

- 这两个方法必须等文档加载好后才能执行，否则会出错。
- insertAdjacentText只能插入普通文本，insertAdjacentHTML插入html代码。
- 使用insertAdjacentHTML方法插入script脚本文件时，必须在script元素上定义defer属性。
- 使用insertAdjacentHTML方法插入html代码后，页面上的元素集合将发生变化。
- insertAdjacentHTML方法不适用于单个的空元素标签(如img, input等)。

案例演示:

```
1 <div id="insert">
2     <p>为书法的人</p>
3 </div>
4
5 <script>
6     var div = document.getElementById("insert");
7     div.insertAdjacentHTML('beforeBegin', 'IIS你家: ');
8 </script>
```

编写一段兼容性代码，用来设置任意标签的文本内容

```
1 <a href="https://www.baidu.com" id="a">打开百度，你就知道! </a>
2
3 <script>
4     var a = document.getElementById("a");
```

```
5     setInnerText(a, "你要打开百度吗? ");
6     console.log(getInnerText(a));
7
8     /*获取任意标签的内容*/
9     function getInnerText(element) {
10         // 判断浏览器是否支持textContent,如果支持, 则使用textContent获取内容, 否则使用
        innerText获取内容。
11         if (typeof element.textContent == "undefined") {
12             return element.innerText;
13         } else {
14             return element.textContent;
15         }
16     }
17
18     /*设置任意标签的内容*/
19     function setInnerText(element, text) {
20         // 判断浏览器是否支持textContent,如果支持, 则使用textContent设置内容, 否则使用
        innerText设置内容。
21         if (typeof element.textContent == "undefined") {
22             return element.innerText = text;
23         } else {
24             return element.textContent = text;
25         }
26     }
27 </script>
```